MADMAN MACHINE

J.S. Hutchison and W.G. Roman

General Electric Research & Development Center
Schenectady, New York 12345

Abstract

A back-end data base machine is discussed in which the back-end is closely coupled to the host system as an intelligent I/O device. The design of the hardware and software is such that the data base disks can be on either the host or back-end computers. The design is motivated by memory size considerations on mini-computer systems and also by cost considerations of large disks. The implementation of such a system on PDP-11s is discussed.

Introduction

MADMAN (Multi-Access Data MANager) [1] is a CODASYL DBTG (network) data base manager which runs on the PDP-11 under the RSX-11M, RSX-11D, or IAS operating systems. MADMAN is designed to run in on-line real-time environments; it has a system level concurrency control to guarantee data base consistency [2] and automatic recovery facilities and other features to maintain the integrity of the data base.

MADMAN is currently being used in a number of General Electric factories along with an associated transaction processor. However after loading the operating system, data manager, transaction manager and journal, etc., little space is left for the application programs. In all PDP-11's except for the PDP-11/70, the maximum memory size of 248k bytes is a limiting factor on the use of the system. An obvious solution for this is to off-load the data manager into a back-end computer.

While back-end computers are not new [3], the traditional implementation has several drawbacks (especially for mini-computer systems): The communication link between the host and back-end has a large overhead and the data base disks are exclusively on the back-end machine.

In order to remedy these deficiencies, a configuration shown in figure 1 is used for the MADMAN machine. The MADMAN machine is designed to be an intelligent I/O device with direct memory access (DMA) to the host computer. Thus, the communication between the host and back-end has low overhead, since I/O operations and commands are optimized for speed. In addition, since the data base machine has direct access to the host memory, it can transfer necessary information when needed, not when the host decides to send it, thus minimizing buffering requirements.

In order to use the disks on the host system, the MADMAN machine has a small cooperating task on the host system which initiates any I/O from those disks to memory which is shared between the disks and the back-end. Use of the host system disks is needed because a major price of mini-computer systems is the disk units and on smaller systems we do not wish to require an extra disk for the data base. However on larger systems, where the data base takes more than one disk, it is expected that the dedicated data base disks will be put on the back-end computer. In this case, the host system will be relieved of the I/O load for those data base disk transfers. Note that even with the larger data bases, some of the data base may reside on the host computer disks.

It should be noted that because of thruput requirements, MADMAN can handle multiple data base commands simultaneously. That is, if disk I/O is necessary during the processing of a data base command, MADMAN causes the I/O to be initiated and then attempts to process other commands. It does not just process one command at a time. This requirement to handle multiple requests simultaneously has an impact on the hardware interface command structure between the host and back-end systems.

The current implementation is for a PDP-11 host and LSI-11 back-end. However, the design allows both of these systems to be changed:

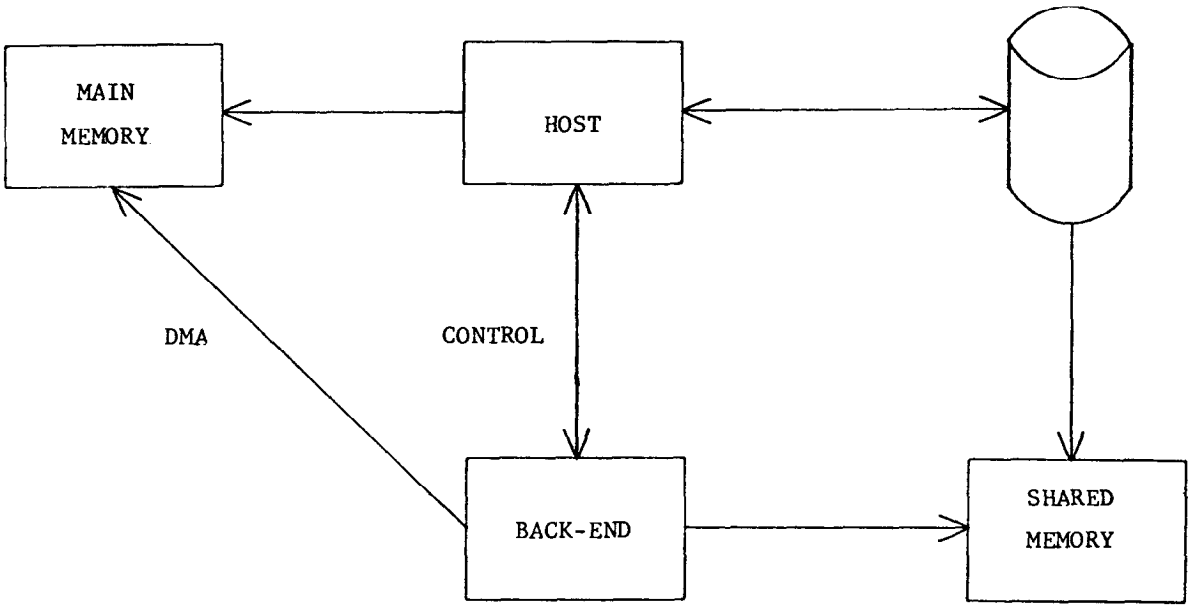

FIGURE 1

One can use more specialized hardware for the back-end, such as described in [4] or possibly just micro coding the current implementation of MADMAN.

The MADMAN back-end can be used with other hosts without having to reimplement the entire data base manager. In fact, the back-end concept allows the use of a data base manager on mini-computers which could not support a data base manager, due to memory size limitations of the host mini-computer.

Other extensions of the basic configuration are possible, such as having multiple hosts using the same back-end, or building a high reliability system with redundant back-ends and hosts, ...

The feature of the traditional implementation that is given up is allowing the host systems to be physically separated by long distances from the back-end. However in giving this up, one improves the response time and keeps the cost of the back-end down.

General Description

Our fundamental concept in designing a back-end data base management machine is an intelligent direct memory access (DMA) peripheral, similar to an IBM/360 I/O channel, implemented using a general purpose computer and special interface hardware. As such, it is very tightly coupled to its host machine; communication between host and back-end is via direct connection to the host's memory and I/O buses.

The general scheme is that the host tells the back-end to start processing and gives the back-end a pointer to a list of command blocks for requests to be processed. The format of command blocks is system dependent, to take advantage of existing system formats and executive structure. Whenever the back-end has completed a request, it interrupts the host to signal the completion of a request and also indicate which request is done; it then continues with other requests, if any. Note that because of I/O delays, the requests may be completed in an order which is different than the order which they were queued to the back-end. This causes no problems because application programs are allowed only one request to the DBMS at a time.

The host adds command blocks to the command list as it receives requests from application programs. The back-end takes requests off of the command list when it is ready to process another request. In order to handle any queuing-dequeuing race condition, when the back-end indicates the completion of a request which it thinks is the last in the command list, it signals this in the completion status. The host must then test whether the request is in fact the end of the list and if it is not then the host must restart the processing of the command list.

When the back-end wants to perform I/O on disks which are on the host system, it interrupts the host to request disk I/O and passes any relevant parameters for the I/O operation. The host performs the *operation into or out of the shared memory* and when it is completed signals the completion to the back-end. The protocol for passing I/O request information and the design of the shared memory system interface is such that more than one I/O can be outstanding at once.

Communications requirements between the host and back-end may be summarized as follows:

Executive level communications for initiating data base access requests and receiving status information on their completion.

Direct memory access to the host machine for interpretation of parameters and commands in the application program and data transfer to and from the *program.*

Access to shared mass storage devices for storing and retrieving information in the data base.

Control and maintenance functions such as halting, starting, resetting, and bootstrapping the back-end processor.

The executive level communications and the control and maintenance functions are analogous to disk controller functions.

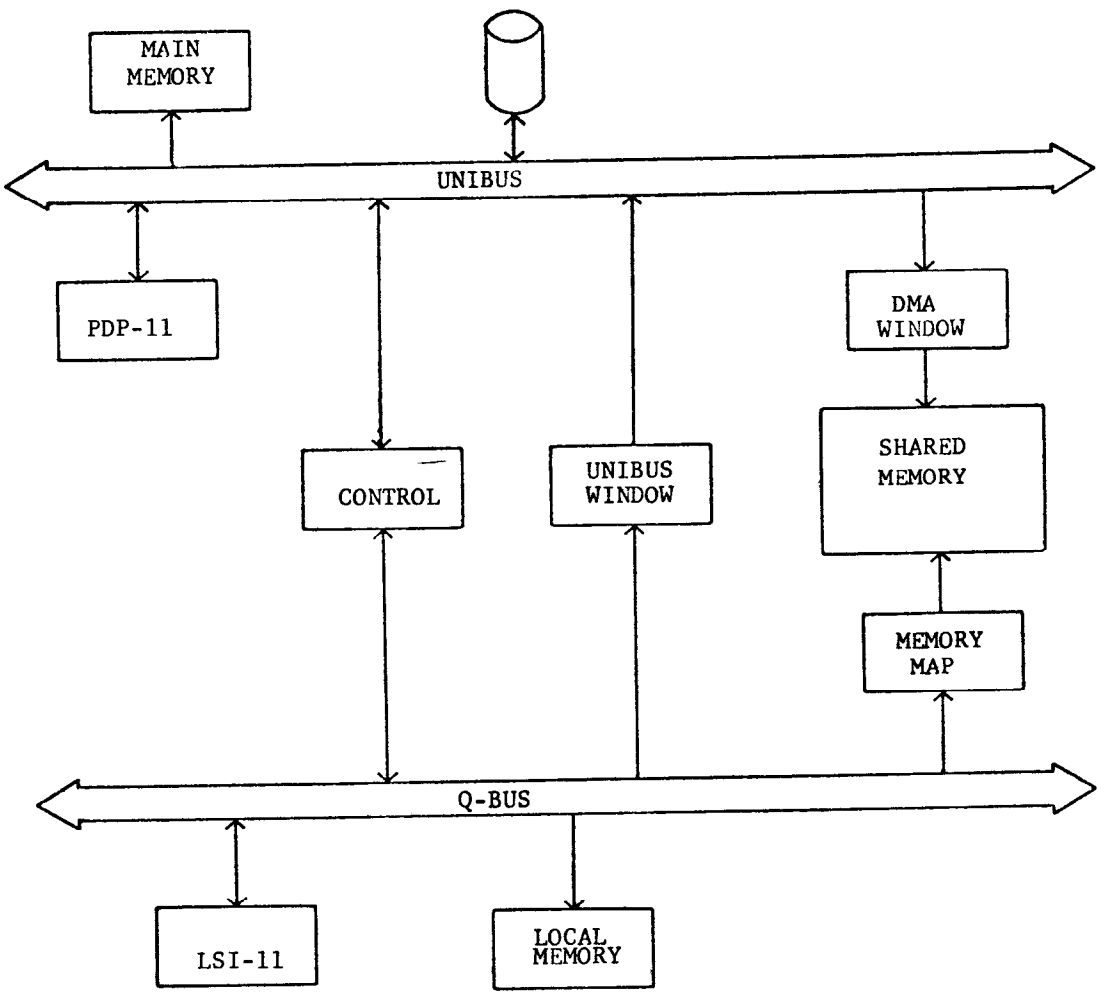

FIGURE 2

They are implemented in a manner similar to accessing a disk drive on the host machine, i.e., via control, status and parameter registers and the interrupt facilities of the host processor.

Direct memory access is implemented as it would be for a disk, by connection to the memory bus of the host and memory cycle stealing.

Access to shared mass storage may be implemented as a shared (dual controller) disk or shared (dual port) memory for I/O buffers, depending on which is more appropriate to the architecture of the host.

Implementation on PDP-11

Figure 2 is a block diagram of a MADMAN machine implementation for PDP-11 with an LSI-11 back-end. This configuration is for all models of the PDP-11 other than the PDP-11/70, which is discussed below. In this initial implementation, all disk storage used for the data base is directly connected to the host; thus, access by the back-end to the data base is through a shared memory subsystem.

The control unit is a dual purpose interface. It is used by the MADMAN software to accept data base access requests and return status on their completion and to request disk I/O operations and receive notification of their completion. It is also used for hardware control of the LSI-11 back-end processor upon system startup and for test and diagnostic purposes.

The control unit consists of two subsections. The console interface provides basic hardware control over the LSI-11; it has the capability to examine and modify memory on the Q-bus, examine and modify internal processor registers, and halt, reset and start the processor. These functions are accomplished through the LSI-11's unique processor console interface. Rather than having the familiar lights and switches, the LSI-11 uses a simple terminal interface for its basic console functions; terminal commands are interpreted by microcode. By replacing the normal serial terminal interface with a software compatible parallel interface to the UNIBUS, all console functions are placed under control of the host PDP-11.

In addition, the console interface provides a means for the host and back-end processors to interrupt each other. When one processor writes a word of data to the console interface, the other processor is interrupted; it then reads the data, which causes an interrupt on the first processor. This interface is also used by the MADMAN software to pass requests and status information between the host and back-end during normal system operations.

For most data base operations the console interface is augmented by a set of mailbox registers. They provide full duplex data transfers between the host and back-end; data written in a mailbox register by one processor may be read by the other processor. The mailbox registers are used for passing additional parameters for a request initiated via the control interface; the control interface interrupt facility is used to synchronize access to the mailbox.

The UNIBUS window allows the back-end to access memory on the host. Segments of Q-bus address space are mapped onto UNIBUS address space, allowing any LSI-11 memory reference instruction to be used on the PDP-11's memory. Note that this is a one-way window; no capability is provided for the PDP-11 to access the Q-bus directly, as this would create a potential for deadlocking one or both buses.

The UNIBUS window is used by the MADMAN software for interpreting data base requests which were initiated via the control interface (similarly to an IBM I/O channel executing a channel program), fetching parameters for a data base request from a user program, and transferring data between a user program and the data base I/O buffer area.

The shared memory is used as a buffer area and cache for data base mass storage I/O. Since the buffer area may be very large, and the LSI-11 can only address 56k bytes of memory, the shared memory is connected to the Q-bus by a memory mapping unit. Similar to the UNIBUS window, the map unit relocates portions of the Q-bus address space to physical memory address space; all of the shared memory may thus be accessed, although not all simultaneously.

The DMA (direct memory access) window enables mass storage I/O devices on the host to communicate with the shared memory. Unlike the UNIBUS window and the memory map, the DMA window cannot simply map a portion of UNIBUS address space to shared memory address space. The UNIBUS architecture allows 8k bytes of peripheral device control register address space; the remainder of the UNIBUS is dedicated to normal memory. To be of any practical value, a mapping system would require many windows (one for each possible concurrent disk I/O request), each large enough for an entire data base page (typically up to 2k bytes). Since what little UNIBUS address space was available was heavily fragmented, a different scheme was used.

The DMA window for an entire disk transfer may be collapsed to a single word location on the UNIBUS at the cost of a slight increase in hardware complexity. PDP-11 disk controllers have the capability to operate in a mode in which they do not increment the UNIBUS address during a transfer; all words transferred are read or written at the same address. By making the memory address translation hardware increment the target address each time a word is accessed, the disk transfer is channeled through a single UNIBUS address into the shared memory. At one UNIBUS location per concurrent disk access, there is more than enough room for the DMA windows required.

Implementation on PDP-11/70

The PDP-11/70 is different from other PDP-11s in that it can have more from 248k bytes of memory. This is done by memory mapping in the cache which uses a 22-bit address rather than an 18-bit address as in other PDP-11s with memory management. Also, the high speed peripherals perform data transfers through high speed controllers on a mass bus which is connected directly to the memory cache. Data transfers through the UNIBUS must use the UNIBUS mapping registers which transform the 18-bit UNIBUS address into 22-bit addresses.

Since disk transfers do not use the UNIBUS but go directly to memory, the shared memory system described above for other PDP-11s will not work. The host memory must be used as the shared memory. Because of the larger PDP-11/70 memories,

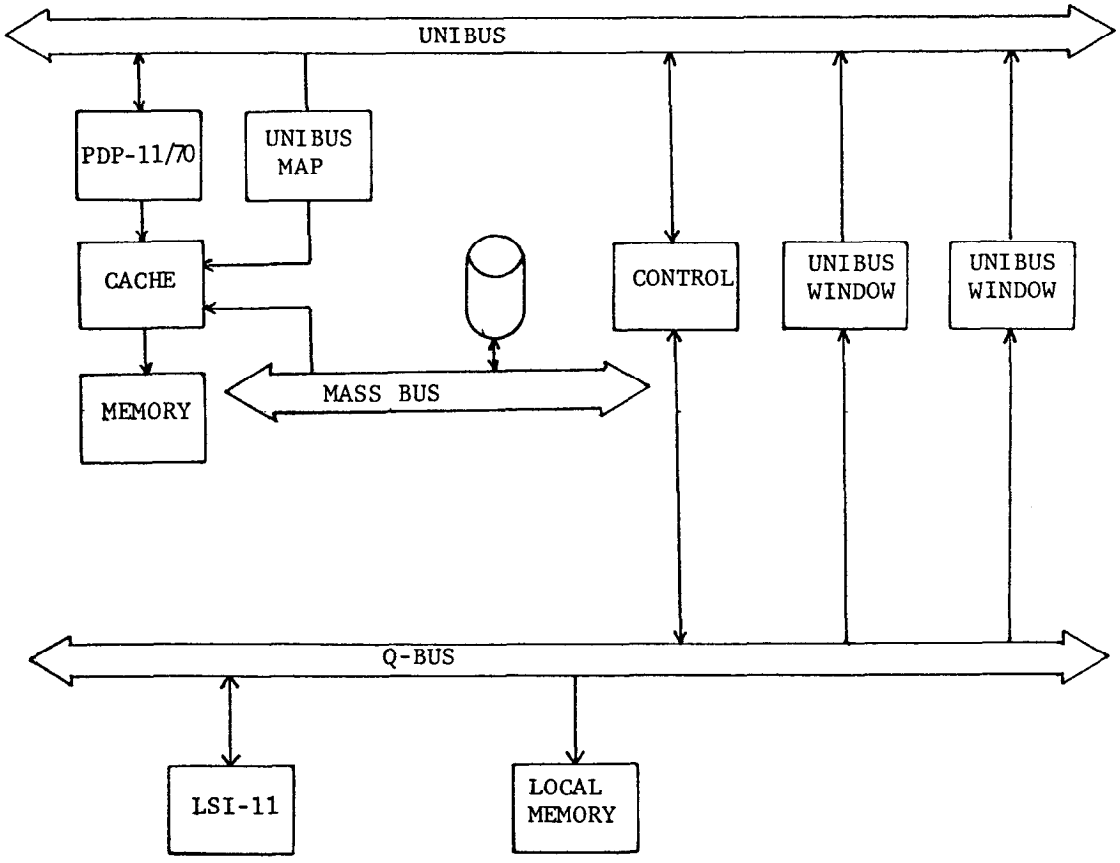

FIGURE 3

this causes no problems. However, because the MADMAN software must be able to access two buffers simultaneously, a second UNIBUS window must be added to the configuration. The PDP-11/70 configuration for the MADMAN machine is shown in figure 3. Figure 3 illustrates a system with the data base disks on the host system. If disks are put on the LSI-11 back-end then the disk I/O would be into or out of the LSI-11 local memory.

Status

The work on the MADMAN machine is now in progress. As of July 1978, the design work has been completed and the hardware and software are being built and tested. The initial version of the MADMAN machine is for a PDP-11/70. This simplifies the initial amount of hardware that has to be built. We expect to have the first version working during the fourth quarter of 1978.

Conclusion

Our review of the initial design indicates that the only significant load changes may be due to additional interrupts. The number of additional interrupts can be alleviated by having a large buffer pool; this will cause less disk I/O, similar to a cache disk. To further reduce the interrupt load, we will have to move the data base disks to the back-end.

In addition to measuring the interrupt load caused by the MADMAN machine, we have to experiment to match the relative speeds of the two machines. Is an LSI-11 fast enough for a PDP-11/70? Or must we use a PDP-11/34 as the back-end? We then need to look at other configurations, such as an LSI-11 host with an LSI-11 back-end, in order to produce more inexpensive factory control systems.

Many systems which depend on high performance will benefit from having a tightly coupled back-end for data base functions. It is particularly useful because it allows mini-computers and micro-computers to be used for functions which would not otherwise be possible, due to memory addressing considerations.

Acknowledgment

In addition to the authors Bruce Bernstein, Jack Bollen, Ulysses Hughes, Jim Kellerman, Donna Phillips, and Lloyd Smith have made substantial contributions to the hardware and software effort in this project.

References

1. General Electric Research & Development Center, MADMAN User Manual, Schenectady, New York 1976.

2. D.J. Rosenkrantz, R.E. Stearns, and P.M. Lewis, System Level Concurrency Control for Distributed Data Base Systems, Proc. of Second Berkeley Workshop on Distributed Data Management and Computer Networks, May 1977.

3. R.H. Canaday, et al., A Back-end Computer for Data Base Management, CACM 17(1974) no. 10, pp. 572-582.

4. P.B. Berra, Data Base Machines, SIGIR Forum, Vol. XII, no. 3, pp. 4-23.